CSEE W4840 Embedded System Design Lab 1

Stephen A. Edwards

Due February 3, 2005

Abstract

Write a C program that counts in decimal on the XSB–300E board. Learn how to compile and run a program, program the FPGA, and use serial communication for debugging.

1 Introduction

The XSB–300E board consists of a large number of peripherals clustered around a Xilinx Spartan XC2S300E FPGA, which has roughly 300 000 raw gates that can be user-programmed into any configuration. For this lab, you will use a configuration we have provided for you, consisting of the Microblaze 32-bit microprocessor, a UART, and a soft peripheral that controls the two seven-segment LEDs on the board and the bargraph LED.

2 A Warning

The XSB–300E boards are very expensive, finicky, and difficult to acquire. Please treat them gently: they are not built to withstand punishment. In particular, I'm sure all you strong Columbia students could break off any given connector after about two or three disconnect-connect cycles. Leave the cables connected.

Like all electronic equipment, they are very sensitive to electrostatic discharge and the floor of the lab is helpfully covered with carpet that tends to create lots of static. You shouldn't need to touch the boards, but if you must, ground yourself first by touching the case of the power supply.

The boards also do not appreciate being dunked in water, Coca-Cola, milk, crumbs, and just about every other sort of foodstuff. Food and drink are prohibited in the lab. If we catch you eating or drinking in the lab, we will make you install all the Xilinx software from scratch before doing the next lab (the nastiest punishment we could think of).

We only have fifteen boards and they have to last for a while. Anybody who breaks a board will be hauled in front of the class and pelted with rotten tomatoes.

3 Hello World

First, get your XSB–300E board to show signs of life by using the canned project we have provided.

1. Log into one of the clients. These Linux machines are named `micro1.ilab.columbia.edu` through `micro15.ilab.columbia.edu`.

2. Create a directory where you'll put this project and `cd` into it. The Xilinx tools create a *lot* of intermediate files (over a thousand for "Hello World") and do not clean up after themselves, so it is important to keep things segregated.

   ```
   $ mkdir lab1
   $ cd lab1
   ```

3. Unpack the template project files by un-tarring them from my directory:

   ```
   $ tar zvxf ~sedwards/4840/lab1.tar.gz
   ```

4. Start the synthesis by invoking make in that directory:

   ```
   $ make download
   ```

   This takes a long time and generates lots of harmless messages and over a thousand temporary files, but will eventually compile the project into a .bit file suitable for the FPGA on the XSB board and finally download it.

   Fortunately, additional invocations of "make download" will only recompile the files that have changed. In fact, use "make download" to run the "hello world" system.

5. Meanwhile, in another window, start `minicom`, a serial communications program. The boards have a serial port on them and the sample project includes a UART (Universal Asynchronous Receiver Transmitter) that can be used to communicate through a null modem cable to the host. We will use this for debugging since it provides a way to print things from the C program.

   Once Minicom is running, make sure it is using the first serial device, `/dev/ttyS0`, and operating at 9600 baud, 8 data bits, 1 stop bit, no parity bits (displayed as `9600 8N1` in the bar along the bottom), and not use hardware or software flow control. "Ctrl-a o" opens a configuration menu.

   Leave Minicom running while you're working on your board: it will display characters printed from the C program running on the board.

6. If all goes well, the project will be compiled, downloaded, and run on the board. The LEDs should flash for a while and "Hello World!" should appear in Minicom. Make sure your board is connected properly (it should have a power connection from the external power supply, a connection through a parallel cable, and a connection through a serial cable) and powered on. An LED near the power connector lights whenever power is applied to the board.

   If you can't get things to work, pester someone for help.

```
Makefile
system.mhs
system.mss
c_source_files/main.c
pcores/clkgen_v1_00_a/data/clkgen_v2_1_0.mpd
pcores/clkgen_v1_00_a/data/clkgen_v2_1_0.pao
pcores/clkgen_v1_00_a/hdl/verilog/clkgen.v
pcores/opb_xsbleds_v1_00_a/data/opb_xsbleds_v2_1_0.mpd
pcores/opb_xsbleds_v1_00_a/data/opb_xsbleds_v2_1_0.pao
pcores/opb_xsbleds_v1_00_a/hdl/vhdl/opb_xsbleds.vhd
data/system.ucf
etc/bitgen.ut
etc/fast_runtime.opt
FILES
```

Figure 1: Files in the "hello world" project.

4 What is going on?

Figure 1 lists the files in the sample project.

The `Makefile` contains the rules for building the project from source. It also contains explicit pathnames to all the tools, which are located in various subdirectories under `/usr/cad`. All the tools require various enviroment variables to be set, but the Makefile takes care of all of this.

The `system.mhs` file lists how the hardware should be assembled to create the project. For example, it lists the LED and serial connections, the Microblaze processor, the UART, and the peripheral that controls the LEDs. Probably the most interesting lines in this file are those that define the LED peripheral:

```
BEGIN opb_xsbleds
 PARAMETER INSTANCE = leds
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFF0200
 PARAMETER C_HIGHADDR = 0xFEFF02ff
 PORT OPB_Clk = sys_clk
 BUS_INTERFACE SOPB = myopb_bus
 PORT RIGHT_LED = RIGHT_LED
 PORT LEFT_LED = LEFT_LED
 PORT BAR_LED = BAR_LED
END
```

This instantiates an `opb_xsbleds` "core" connected to the main processor bus and memory mapped from addresss `0xFEFF0200` to `0xFEFF02ff`. These magic numbers will be useful later when you write code that controls the LEDs.

The `system.mss` file describes how the software should be assembled for the system, and is less interesting than `.mhs` file. It says to include the UART driver and to connect stdin and stdout of the program to the driver.

The `main.c` file contains the `main()` function that is run when the system is downloaded. You will modify this for this lab.

The files under `pcores/opb_xsbleds_v1_00_a` describe the LED peripheral. The `.mpd` file describes the interface and configuration parameters of the peripheral. The `.pao` file lists the VHDL source files for the peripheral.

The real action happens in the `opb_xsbleds.vhd` file, which describes the LED peripheral in detail. It lists the interface to the OPB bus and describes a pair of processes, one that determines when to respond to bus requests (i.e., an address decoder) and one that copies bits from the data bus to registers controlling the LEDs when active.

bits								address
rdp	rg	rf	re	rd	rc	rb	ra	0xFEFF0200
ldp	lg	lf	le	ld	lc	lb	la	0xFEFF0204
b7	b6	b5	b4	b3	b2	b1	b0	0xFEFF0208
						b9	b8	0xFEFF020C

Figure 2: The LED registers. a–g and dp are the LED segments and decimal point; b0–b9 are the bargraph segments.

The files under `pcores/clkgen_v1_00_a` describe a core that drives the processor's clock from the clock on the XSB board.

The `data/system.ucf` lists to what pins on the FPGA internal signals should be connected. These pin numbers came from the XSB board documentation, which lists the role(s) of each pin.

Finally, `bitgen.ut` sets some options for generating the final bitstream, `fast_runtime.opt` is a script that controls the Xilinx XST synthesis software, and `FILES` lists the files that are part of the project, which makes it easy to create new .tar.gz files.

5 The LED Peripheral

The OPB-XSBLEDs peripheral is simple. It consists of four eight-bit memory-mapped write-only registers that control the two seven-segment displays and the bargraph display. The layout of these registers are shown in Figure 2.

6 The Assignment

Modify the hello.c file to count in decimal from 0 to 99 on the LEDs. Make sure the numbers from 0–9 don't display a leading 0. Make the bargraph display the least significant digit, i.e., light the lowest bar when the least significant digit is a 0 and the highest when it is 9. Write the following things:

- A decimal-to-seven-segment decoder that transforms the numbers 0–9 into appropriate bit patterns for the LEDs
- A decimal-to-bargraph decoder that transforms the numbers 0–9 into bit patterns for the bargraph
- A counting loop that feeds 0–99 to the LEDs in sequence

Code this in C by modifying `main.c`.

Show your working counter to a TA, have him sign a printout of your solution (i.e., `hello.c`), and hand that in.

Shorter, elegant, and readable solutions will be scored higher than ones that merely work.